H() WRAPPER

`h()` wrapper

VIRTUAL-HYPERSCRIPT

A DSL for creating virtual trees

MANY IMPLEMENTATIONS

- Ours: https://gist.github.com/whiteinge/02a4868ff91aeacf3b99
- react-hyperscript: https://github.com/mlmorg/react-hyperscript
- r-dom: https://github.com/uber/r-dom
- Or, maybe: `var h = React.createElement.bind(React);`

COMPARE AND CONTRAST W/ JSX

```
return (
    <ul>
        <li>foo</li>
        <li>bar</li>
    </ul>
);

// vs

return (
    h('ul', [
        h('li', 'foo'),
        h('li', 'bar'),
    ])
);
```

REACT.CREATEELEMENT()

```
// React.createElement('element', {attributes}, [children]);

React.createElement('ul', {
    className: 'someclass',
}, [
    React.createElement('li', null, ['foo']),
]);
```

`h()` IS A SHORTCUT

```
React.createElement('ul', {
    className: 'someclass',
}, [
    React.createElement('li', null, ['foo']),
]);

// vs.

h('ul.someclass', [
    h('li', 'foo'),
]);
```

INDENT `H()` EXACTLY LIKE WITH HTML

```
<div>
    <ul>
        <li>foo</li>
        <li>bar</li>
    </ul>
    <p>Para Stuff</p>
</div>

// and

h('div', [
    h('ul', [
        h('li', 'foo'),
        h('li', 'bar'),
    ]),
    h('p', 'Para stuff'),
]);
```

ADD ID AND CLASS INLINE

```
<div id="foo" class="bar">Foo</div>

// vs.

h('div#foo.bar', 'Foo');
```

USE CUSTOM HTML ATTRIBUTES LIKE WITH HTML

```
<div
        foo="Foo"
        bar="Bar"
        baz="Baz"
    >
    Content here.
</div>

// and

h('div', {
        foo: 'Foo',
        bar: 'Bar',
        baz: 'Baz',
    },
    'Content here.');
```

BUT `H()` IS JAVASCRIPT

```
h('ul', arrayOfStuff.map(x => h('li', x)));

// and

h('table.ss-table', [
    h('thead',
        h('tr', visibleGrains.map(x =>
            h('th.search-header', searchHeader(x))))),

    h('tbody', grains.map(mgrains =>
        h('tr', mgrains.map(gval =>
            h('td', gval))))),
    ]);
```

USE JAVASCRIPT VARIABLES WITH `H()`

```
var color = 'red';

h('p', {
    className: color,
}, 'I am red.');
```

Watch out for JavaScript reserved words.

USE `H()` WITH REACT COMPONENTS

```
import {MyComponent} from './components';

h(MyComponent, {props: 'here'});
```

COMPONENTS

- Complex.
- Verbose.
- Stateful.
- Great for encapsulating *private* state or making advanced use of lifecycle methods.

COMPONENTS

```
// Creation
var MyComponent = React.createClass({
    propTypes: {
        ...,
    },
    function lifecycleStuffs() {
        ...,
    },
    function someHelperMethod() {
        ...,
    },
    function render() {
        return h('p', 'stuff');
    },
});
```

STATELESS FUNCTIONAL COMPONENTS

```
var MyComponent = function(props) {
    return h('p', 'A component!');
};

h(MyComponent);
```

AND DON'T OVERLOOK THE HUMBLE FUNCTION

```
var assembleAWhole = function(part1, part2) {
    return h('div', [
        part1,
        part2,
    ]);
};

// example

var foo = h('p', 'foo');
var bar = h('p', 'bar');
var vtreeMarkup = assembleAWhole(foo, bar);
```

ESCALATE TO MORE COMPLEXITY AS NEEDED

- Do you need to...
    - Output straightforward, possibly nested markup? `h()`
    - Combine different bits of markup, possibly dynamically? `function`
    - Want a reusable HTML element? `Stateless function component`
    - Want to abstract away complicated markup behind a callable interface? `Stateless function component`, or `function`.
    - Need internal state tracking or lifecycle hooks? `component`